

Temper

Temporal Programmer: An Introduction

Abbas K. Zaidi
Alexander H. Levis
<szaidi2>, <alevis>@gmu.edu

Adversary Behavioral Modeling
Maxwell AFB, Montgomery AL
March 18 - 19, 2008

- **A Logic for Time**
 - **Point-Interval Logic**
 - **Point Graphs**
- **Temper** - Software Implementation of Point-Interval Formalism
- **Temporal Issues in Forensics**
- **Example: Applying Temper to London Bombing Data**

Point-Interval Logic (PIL)



- Allen introduced Interval Algebra as a framework for temporal reasoning. The algebra takes time intervals to be primitives.
- Zaidi expanded this logic by adding points to create Point Interval Logic (PIL)
 - Three Cases
- Case I: X and Y both intervals with non-zero lengths

$X = [sx, ex]$, $Y = [sy, ey]$ with $sx < ex$ and $sy < ey$

Before	$X < Y$	$ex < sy$	
Meets	$X m Y$	$ex = sy$	
Overlaps	$X o Y$	$sx < sy < ex < ey$	
Starts	$X s Y$	$sx = sy, ex < ey$	
During	$X d Y$	$sx > sy, ex < ey$	
Finishes	$X f Y$	$sy < sx, ey = ex$	
Equals	$X = Y$	$sx = sy, ex = ey$	

Point-Interval Logic



Case II: X and Y both

points

$X = [px]$ and $Y = [py]$ with $sx = ex = px$ and $sy = ey = py$

Before	$X < Y$	$px < py$	$\begin{array}{c} X \\ px \\ \bullet \end{array} \quad \begin{array}{c} Y \\ py \\ \bullet \end{array}$
Equals	$X = Y$	$px = py$	$\begin{array}{c} [X;Y] \\ \bullet \end{array}$

A point-point relation “less than or equal to” (\leq) can be added to PIL without losing tractability.

Case III—X is a point and Y is an

interval

$X = [px]$ and $Y = [sy, ey]$ with $px = sx = ex$ and

$sy < ey$

Before	$X < Y$	$px < sy$	$\begin{array}{c} X \\ \bullet \end{array} \quad \begin{array}{c} Y \\ \text{---} \end{array}$
Starts	$X \text{ s } Y$	$px = sy$	$\begin{array}{c} X \\ \bullet \end{array} \quad \begin{array}{c} Y \\ \text{---} \end{array}$
During	$X \text{ d } Y$	$sy < px < ey$	$\begin{array}{c} X \\ \bullet \end{array} \quad \begin{array}{c} Y \\ \text{---} \end{array}$
Finishes	$X \text{ f } Y$	$px = ey$	$\begin{array}{c} X \\ \bullet \end{array} \quad \begin{array}{c} Y \\ \text{---} \end{array}$
Before	$Y < X$	$ey < px$	$\begin{array}{c} Y \\ \text{---} \end{array} \quad \begin{array}{c} X \\ \bullet \end{array}$

- **Quantitative Temporal Information**

- $d1 \leq \text{Length} [X, Y] \leq d2$

- $t1 \leq \text{Stamp} [X] \leq t2$

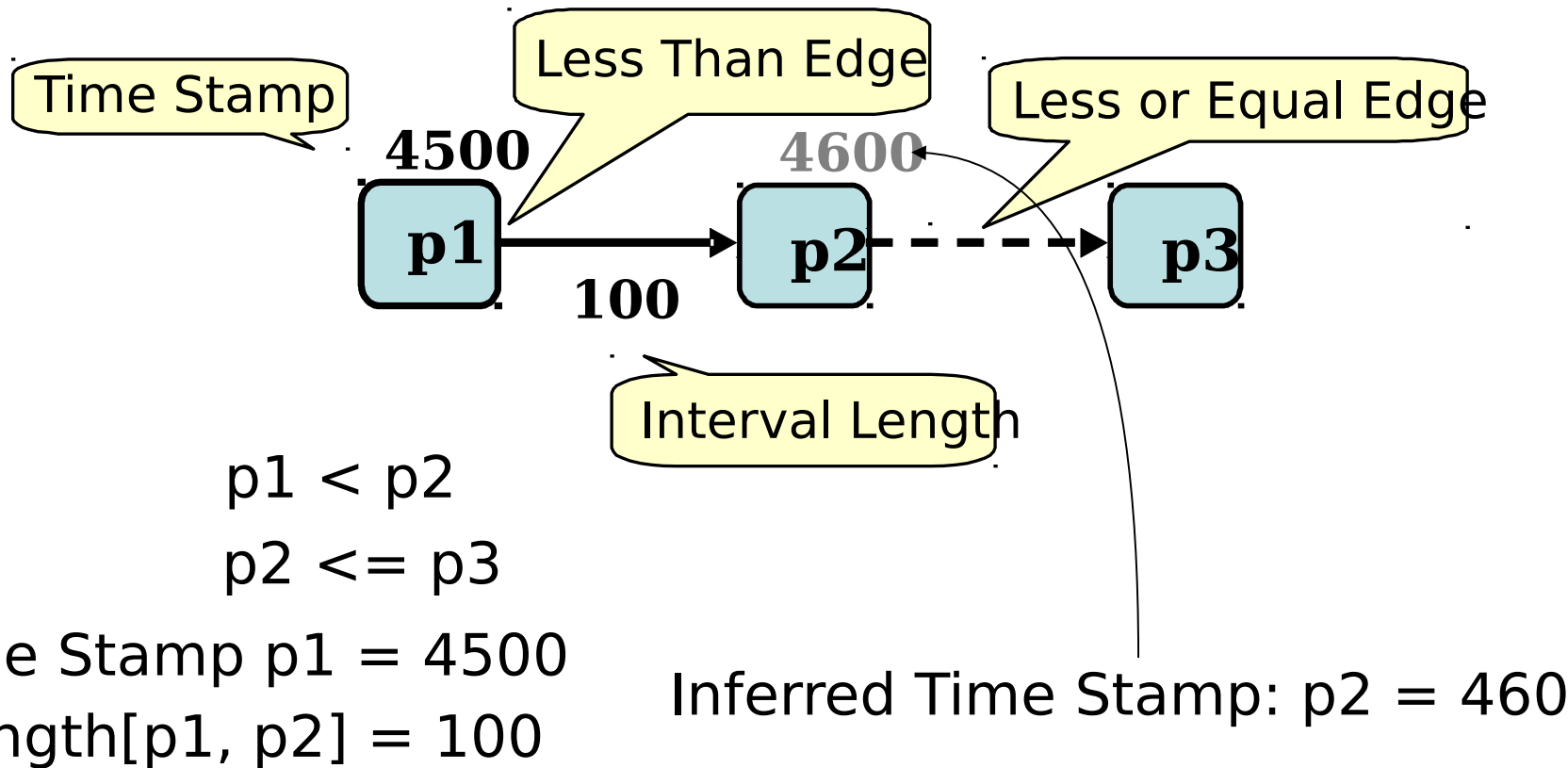
where $d1$, $d2$, $t1$, and $t2$ are rational numbers, and X , Y are points

- This allows for “at least” and “at most” temporal relationships for interval lengths, and “no earlier than” and “no later than” temporal relationships between time points

- **Knowledge Representation**

- A graph with nodes representing time points and edges representing the ‘inequalities’ captures the information in PIL statements

Point Graphs



Point Interval Logic Statements and the corresponding Point

```
sB < eB
sC < eC
sD < eD
sF < eF
sG < eG
sH < eH
Length[sB,eB] = 8
Length[sC,eC] = 9
Length[sD,eD] = 4
Length[sF,eF] = 1
```

Point Graph

Input Window


LT Edge

Time Stamp

LE Edge

Output Window

System Architectures Laboratory
George Mason University



Temper
vikings.gmu.edu

Output	
Earliest Start Time	Latest End Time
0	8
0	12
8	12
9	13
12	15
10	15

Temper - The Software



- **Temper** is a tool for temporal knowledge representation, reasoning, and planning using **Point-Interval Logic (PIL)**.
- **PIL** is a formal algebraic framework for reasoning with time. It has the ability to handle both:
 - Events and Activities
 - Quantitative and Qualitative temporal relationships
 - Reasoning and Planning
- The relationships among various activities and events in a domain are specified in the form of **PIL** statements. These statements are converted into a graphical construct called **Point Graphs (PG)**.
 - Algorithms for verification, inference, and planning are implemented on the Point Graph representation.
- The implementation of **PIL** is in the form of a .NET class library called **PIL Engine**. It provides an application programming interface (API) that can be used in any .NET compliant programming language. It uses **QuickGraph**, which is an open-source C# implementation of the **Graphviz** library from AT&T.
- **Temper** provides a graphical user interface (GUI) to **PIL Engine**.

Add/Delete PIL Statements

Add Stamp Delete Stamp [] = []

Add Length Delete Length [] [] = []

Add Relation Delete Relation [] < []

Add Composite Relation [] < [m] []

Reference Date and Time Wednesday, October 11, 2006 24:00

☐ Numeric Stamp

☒ Day ☒ Hour ☒ Minute ☐ Second

Language
Editor

Query
Editor

Query

Query

Query Stamp []

Query Length [] []

Query Relation [] []

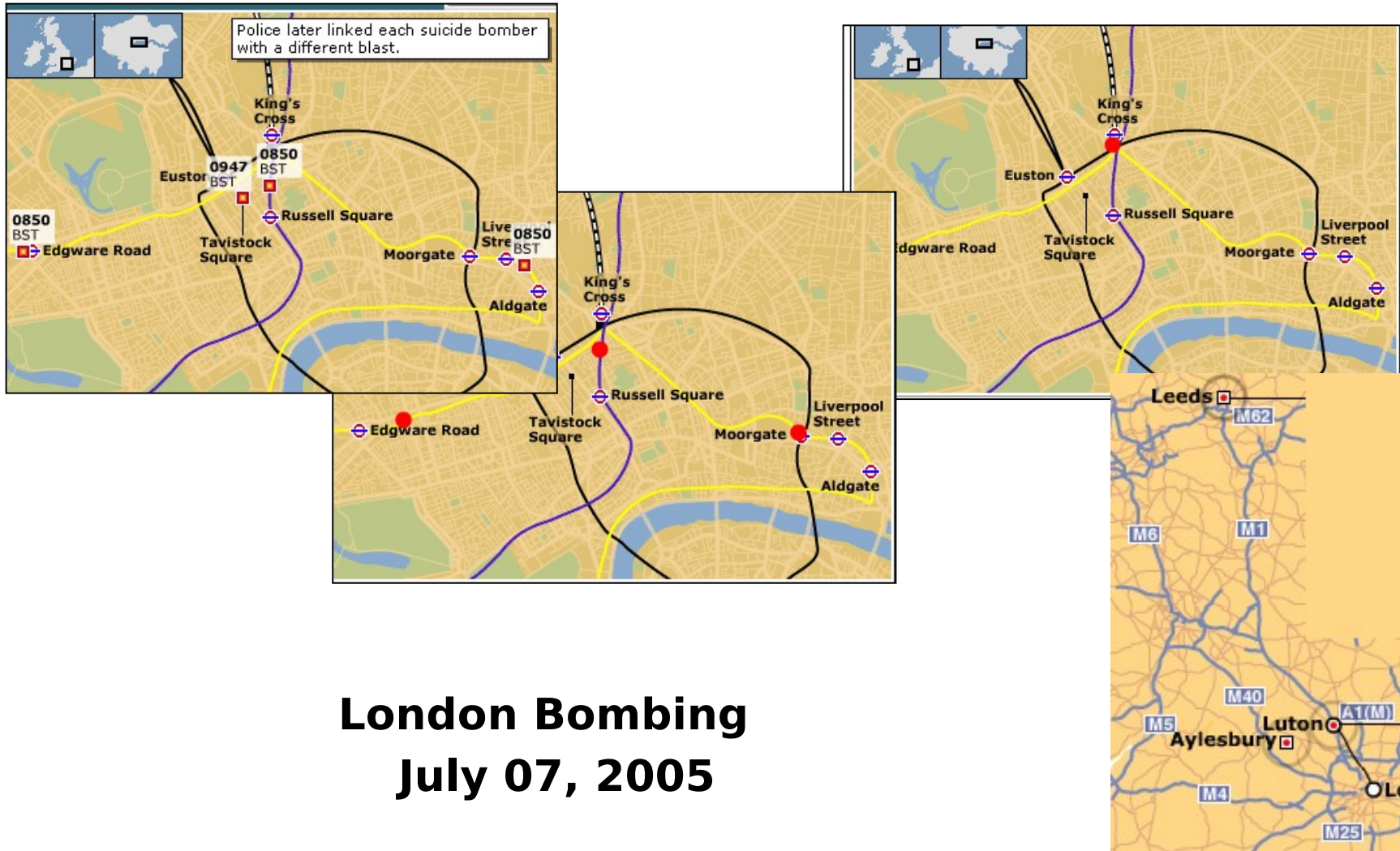
Close

Modeling with **Temper**



- Convert the available temporal information into statements in Point-Interval Logic.
- Input these statements to **Temper** using the language editor of **Temper**.
- Construct a Point Graph representation of the set of Point Interval Logic (PIL) statements.
 - If the set of PIL statements is inconsistent then **Temper** will not be able to construct the Point Graph representation.
 - **Temper** will identify the subset of PIL statements causing the inconsistency.
 - User will remove the inconsistent statements.
- Once a consistent Point Graph has been constructed, it can be used to draw inferences.

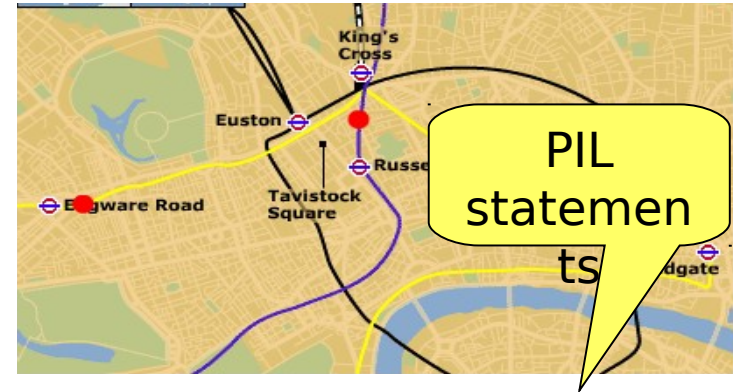
- **Knowledge Management and Reasoning**
 - **Forensics**
 - Understanding of an incident of interest or a critical activity requires reconstruction of events that lead to an observable effect
 - Information regarding the incident/activity unfolds in no specific order and originates from different locations
 - Temporal information may be both qualitative and quantitative
 - Information may be inconsistent/incorrect
 - Information may contain hidden patterns or temporal relations that can help identify missing links
 - This calls for an automated tool for temporal knowledge representation, management, verification and reasoning
- **Temper** is also the temporal algorithm embedded in *Pythia*



Example: London Bombing



- There were four explosions in London.
- The sites of these explosions were: Travistock Square, Edgware Road, Aldgate and Russell Square.
- Three of these explosions (Edgware, Aldgate and Russell Square) were in trains.
- These trains left from King's Cross station. The journey of these trains ended in explosions.
- The time it takes a train from King's Cross to reach Edgware is at least 5 minutes.
- The time it takes a train from King's Cross to reach Aldgate is at least 4 minutes.
- The time it takes a train from King's Cross to reach Russell Square is at least 5 minutes.



Interval Train_King_Cross_to_Edgware,
Train_King_Cross_to_Aldgate,
Train_King_Cross_to_Russell_Sq

Point Explosion_at_Travistock_Square,
Explosion_near_Edgware,
Explosion_near_Aldgate,
Explosion_near_Russell_Sq

Explosion_near_Edgware **finishes**
Train_King_Cross_to_Edgware

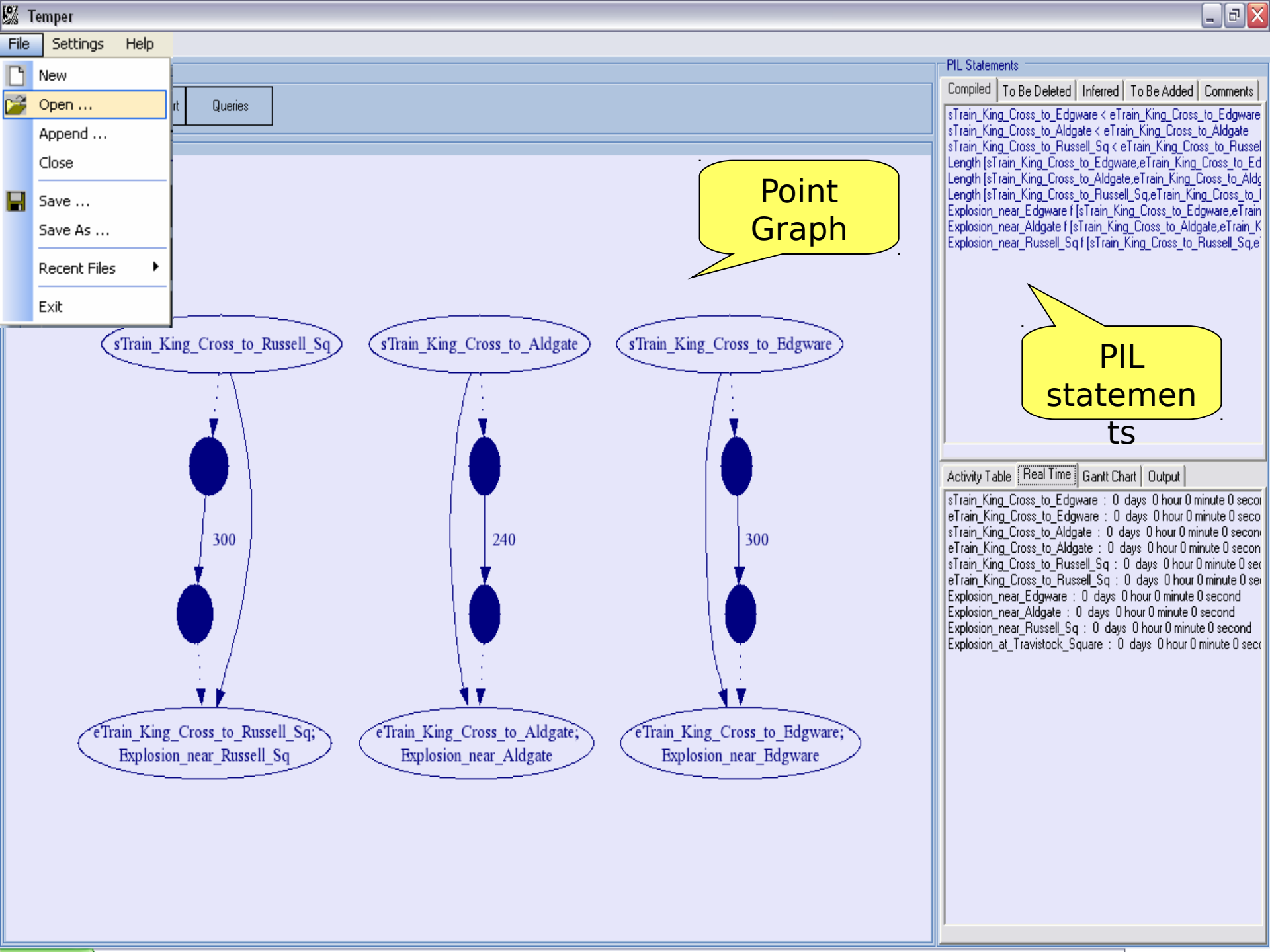
Explosion_near_Aldgate **finishes**
Train_King_Cross_to_Aldgate

Explosion_near_Russell_Sq **finishes**
Train_King_Cross_to_Russell_Sq

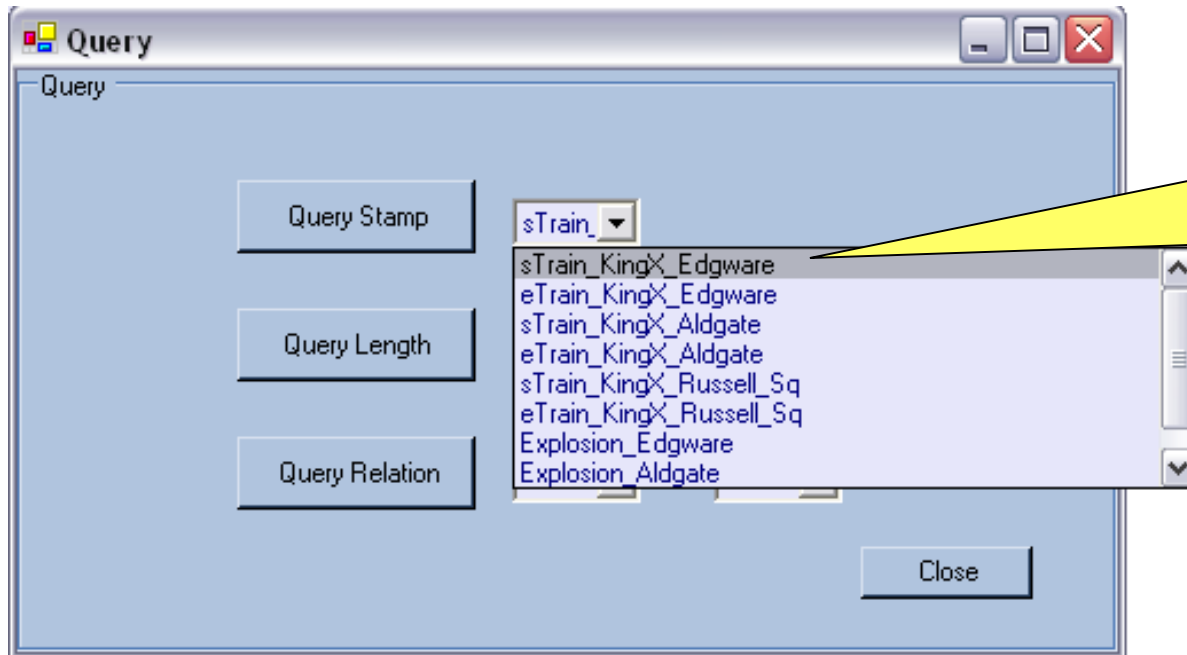
Length [Train_King_Cross_to_Edgware] >= 0:5:0

Length [Train_King_Cross_to_Aldgate] >= 0:4:0

Length [Train_King_Cross_to_Russell_Sq] >= 0:5:0



Example: London Bombing (cont'd)



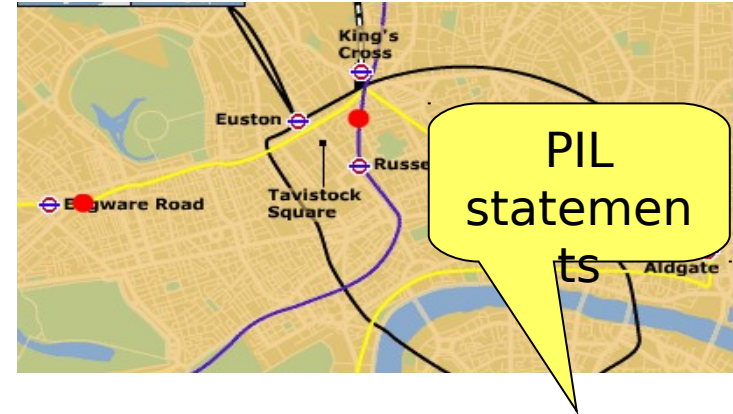
Query Stamp
(when did the
train to Edgbare
leave from King's
Cross?)



Example: London Bombing (cont'd)

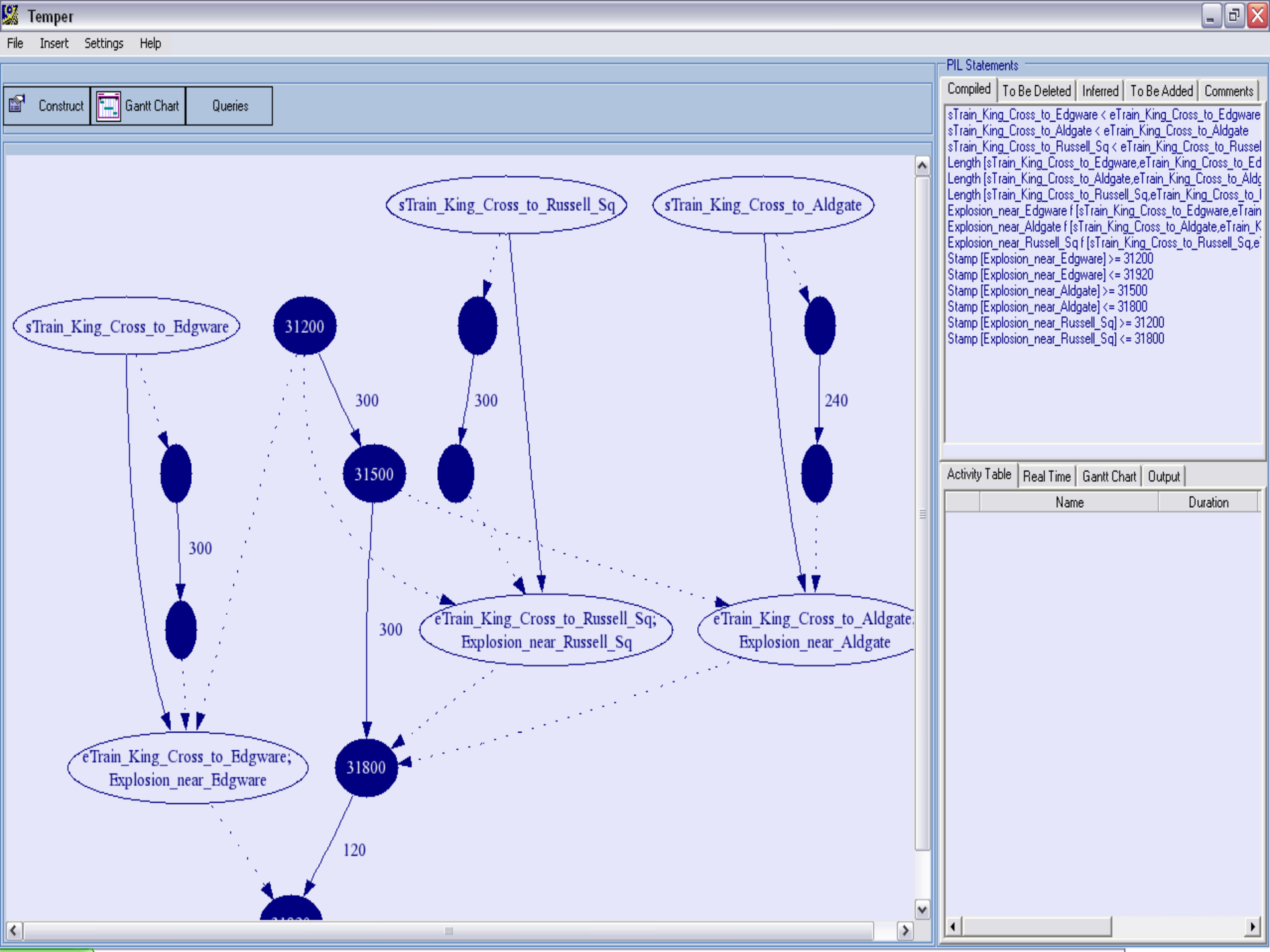


- The explosion near Edgware Road took place between time units 8:40 and 8:52.
- The explosion near Aldgate took place between time units 8:45 and 8:50.
- The explosion near Russell Square took place between time units 8:40 and 8:50.
- The explosion at Travistock Square took place between time units 9:45 and 9:55.

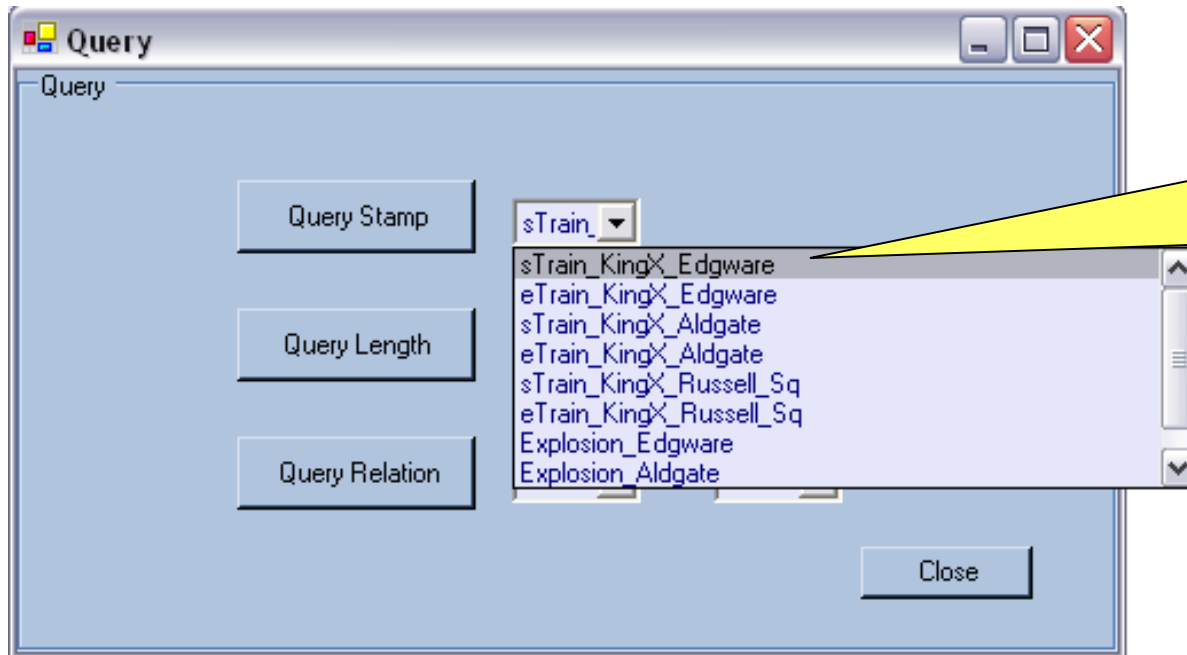


8:40 <= *Stamp* [Explosion_near_Edgware] <= 8:52
 8:45 <= *Stamp* [Explosion_near_Aldgate] <= 8:50
 8:40 <= *Stamp* [Explosion_near_Russell_Sq] <= 8:50
 9:45 <= *Stamp* [Explosion_at_Travistock_Square] <= 9:55

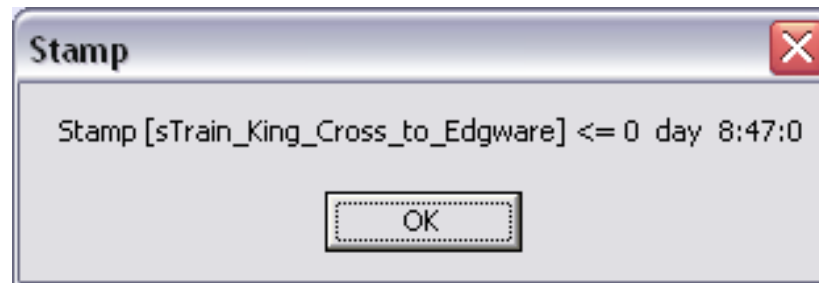


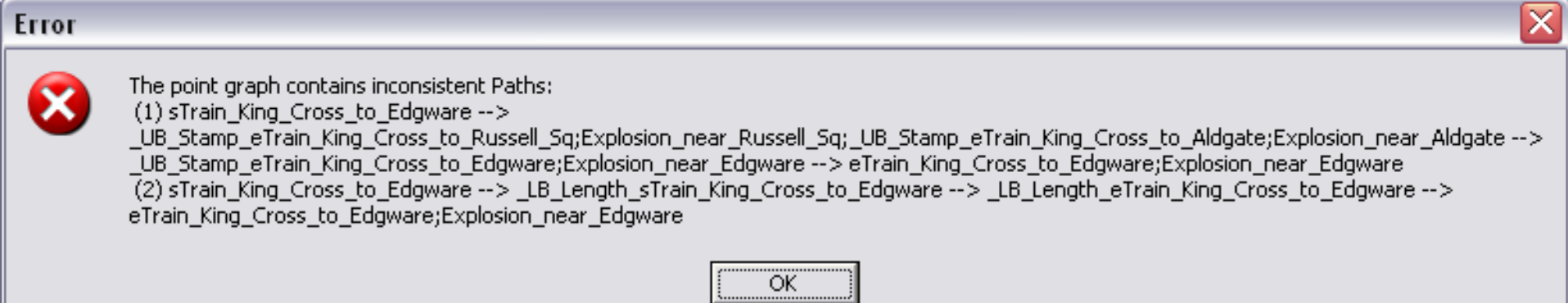
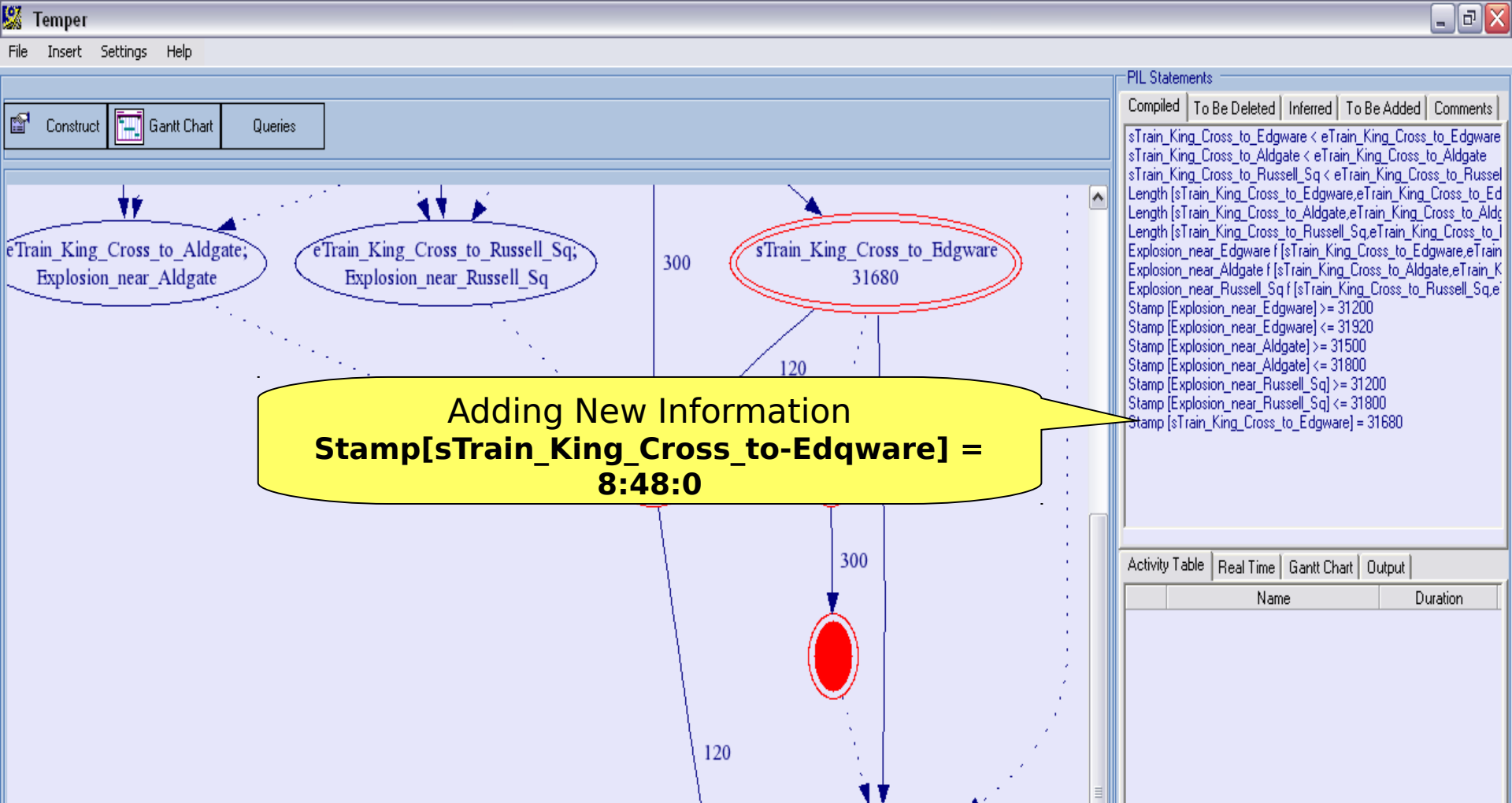


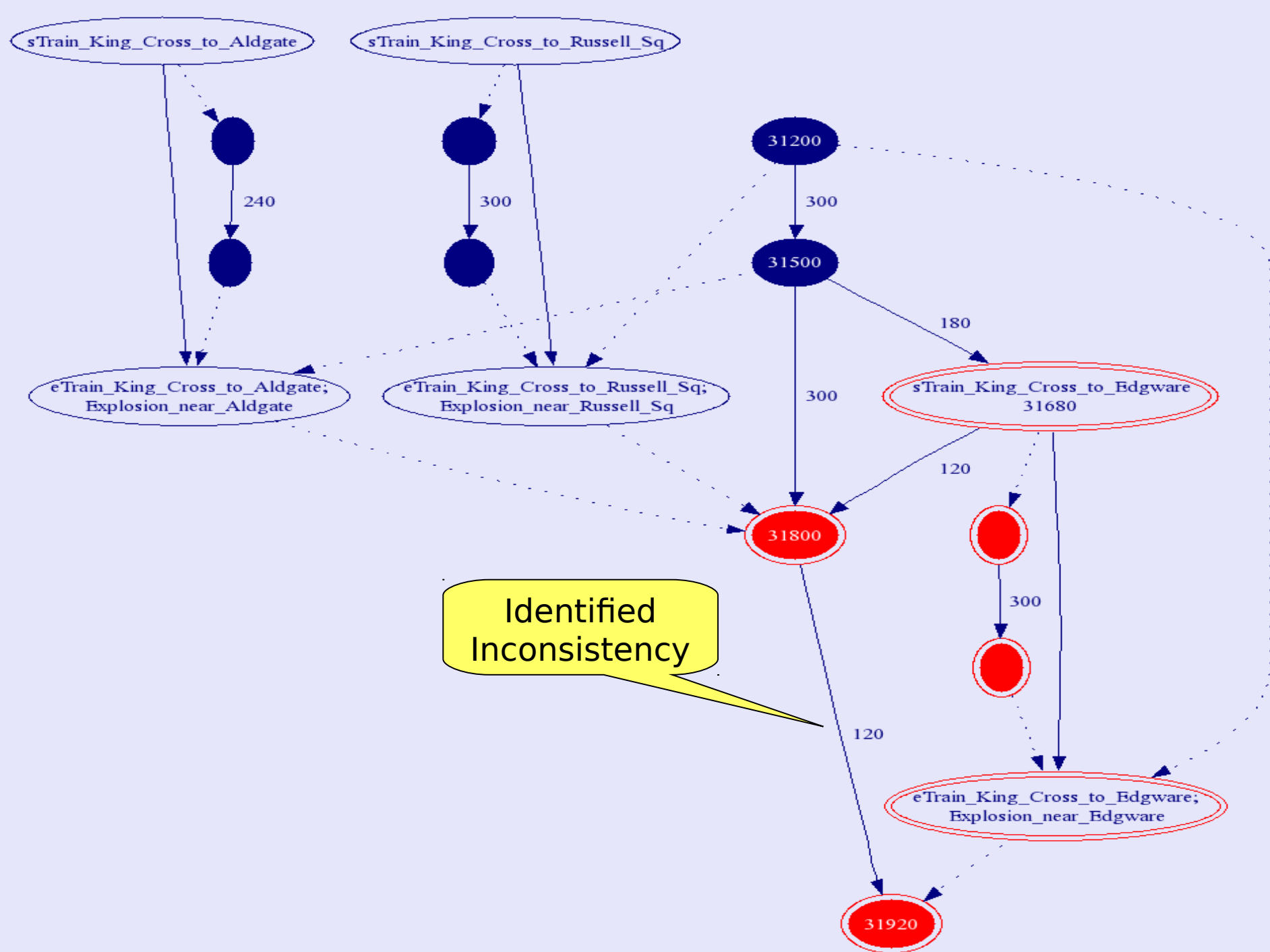
Example: London Bombing (cont'd)

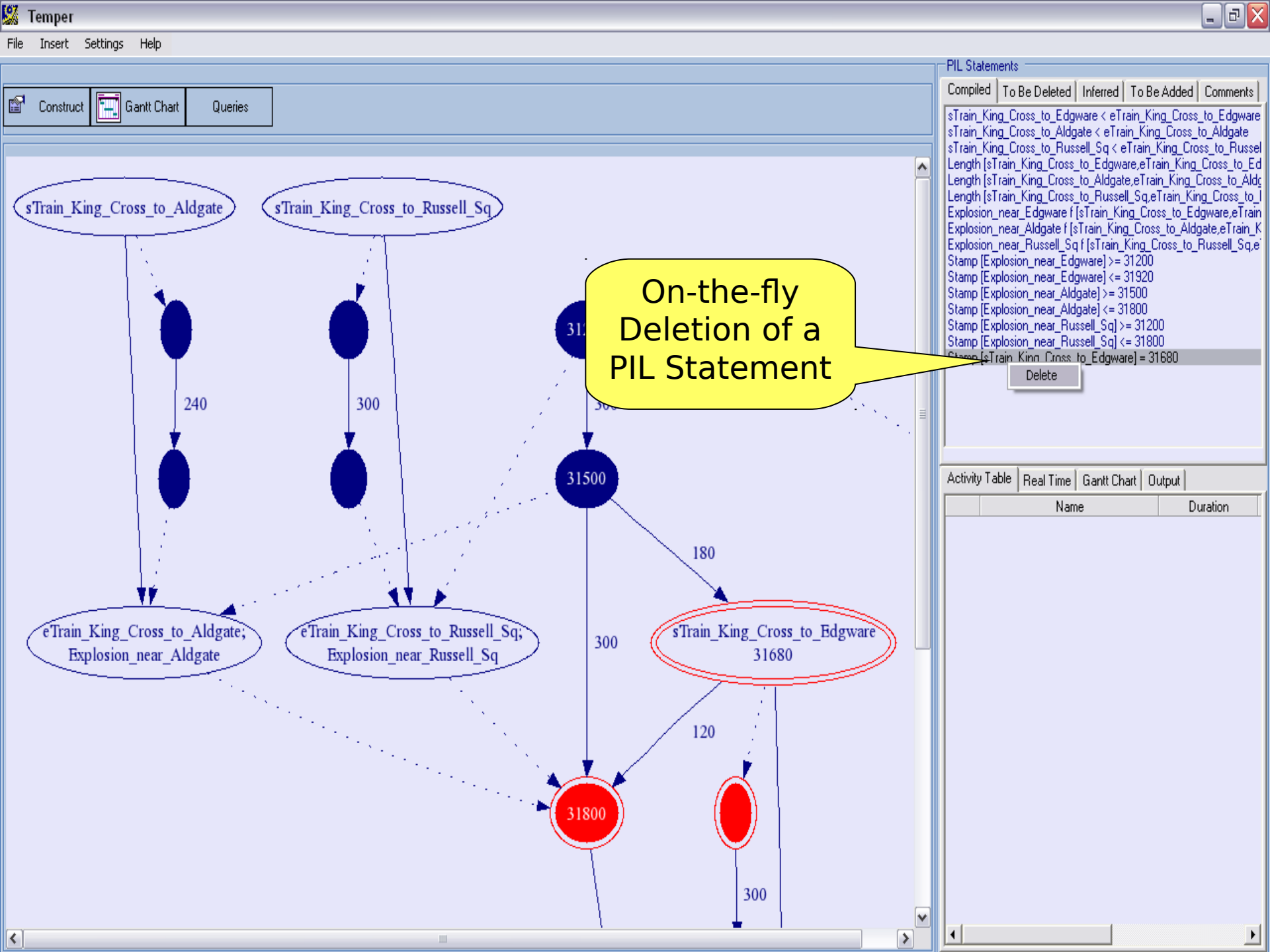


Query Stamp
(when did the
train to Edgbare
leave from King's
Cross?)





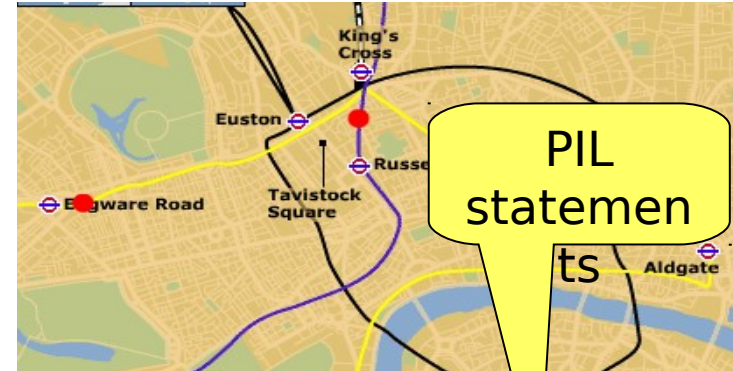




Example: London Bombing (cont'd)



- The alleged four bombers spotted entering the Luton station at time unit 7:20.
- The next train from Luton to King's Cross left at 7:48 reaching King's Cross at 8:42.
- Train to Edgware left after the train from Luton.
- Train to Aldgate left after the train from Luton.
- Train to Russell Sq. left after the train from Luton.



Interval Train_Luton_to_King_Cross

Point Bombers_spotted_at_Luton

Stamp [Bombers_spotted_at_Luton] = 7:20

Stamp [sTrain_Luton_to_King_Cross] = 7:48

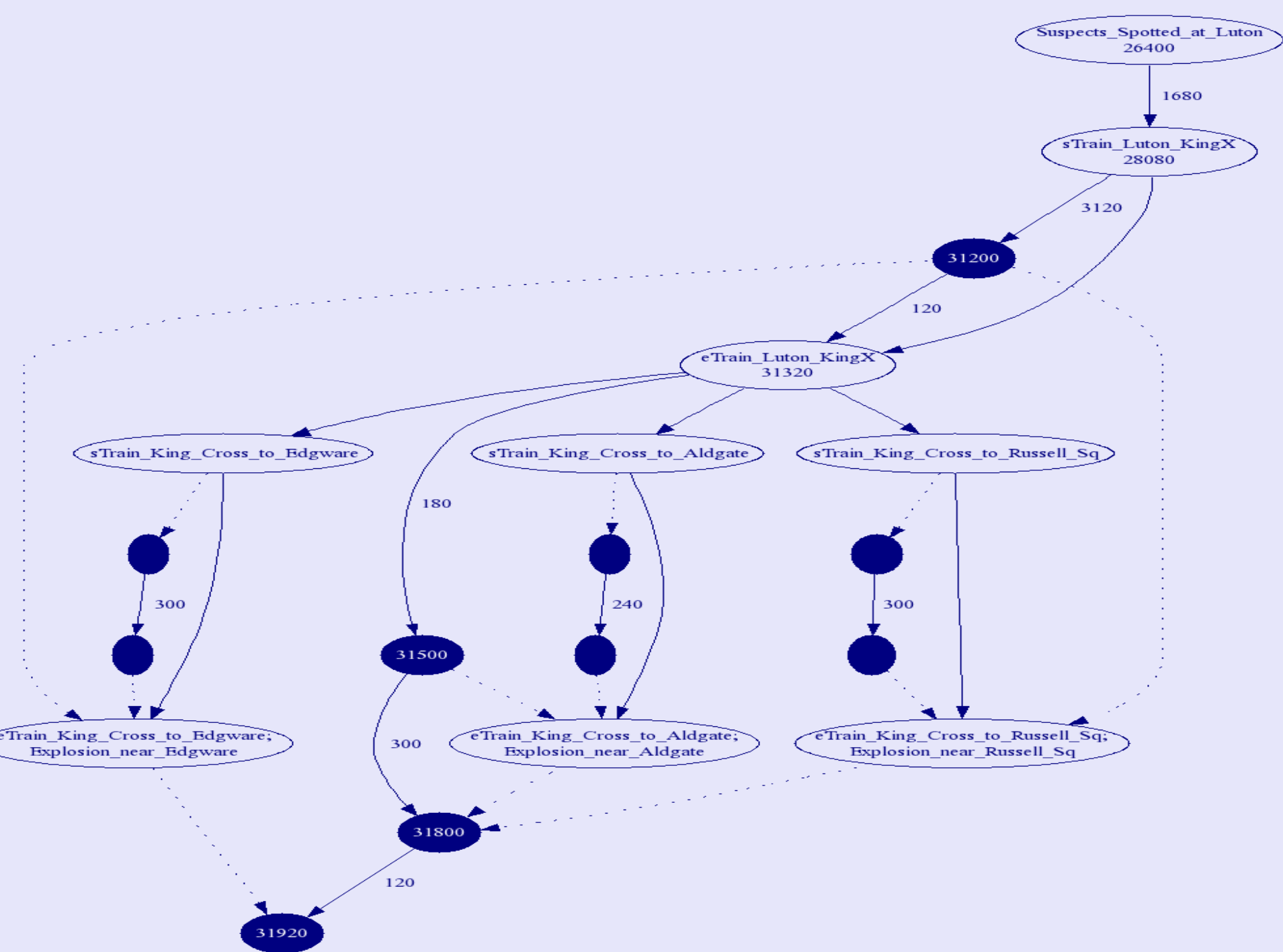
Stamp [eTrain_Luton_to_King_Cross] = 8:42

eTrain_Luton_to_King_Cross *before* Train_King_Cross_to_Edgware

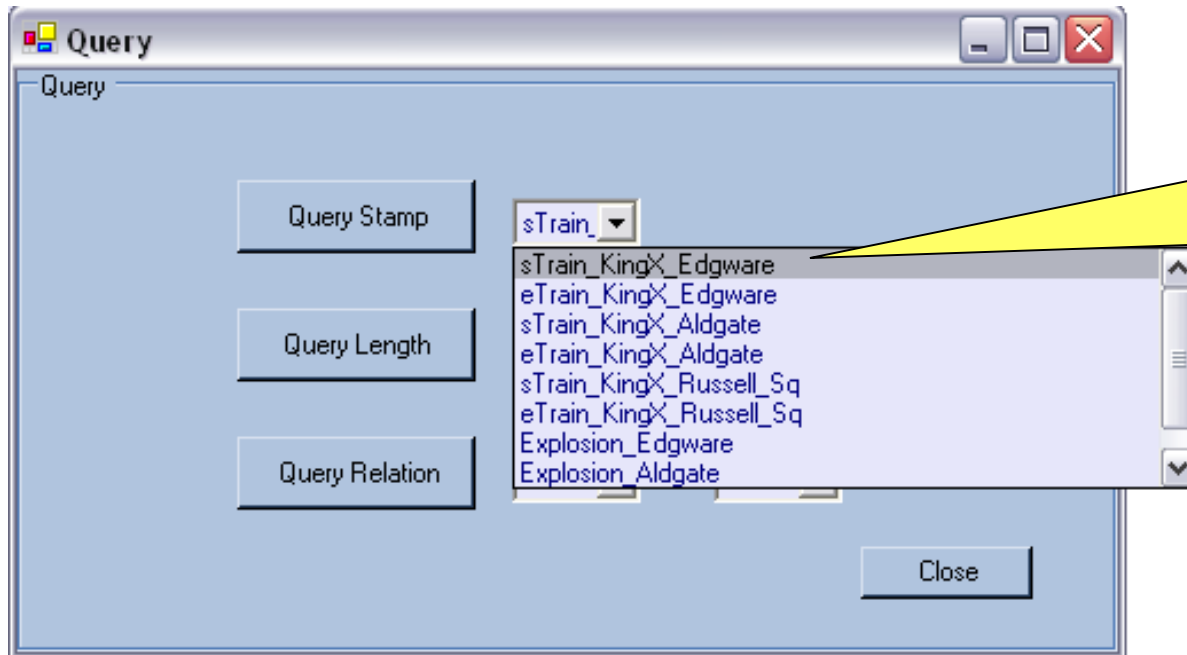
eTrain_Luton_to_King_Cross *before* Train_King_Cross_to_Aldgate

eTrain_Luton_to_King_Cross *before* Train_King_Cross_to_Russell_Sq

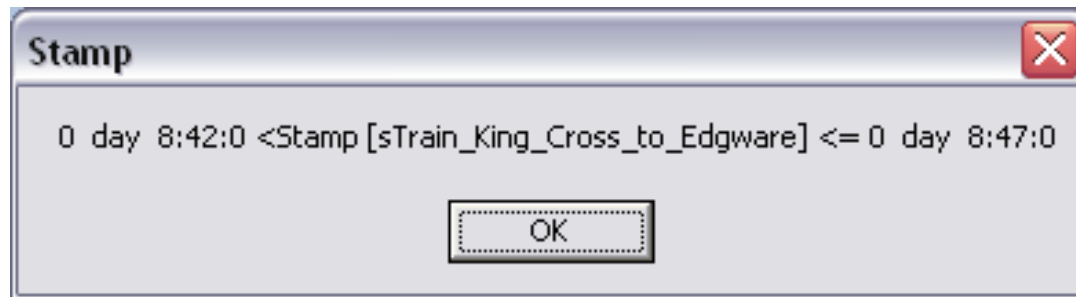




Example: London Bombing (cont'd)



Query Stamp
(when did the
train to Edgbare
leave from King's
Cross?)



Conclusion

- **A formal approach to modeling and analyzing temporal information related to an event of interest, e.g., terrorist acts**
- **A software implementation of the approach with**
 - **An easy-to-use input language**
 - **Analysis toolkit that includes a consistency checker and a reasoning tool with a query language/interface**
 - **An efficient revision mechanism that helps add/modify temporal information without restarting the whole process**
 - **A graphical interface**
- **What might be added in future**
 - **Connectivity to temporal information in databases**
 - **Automated extraction of temporal information from textual source(s)**
 - **Better user/analyst input/output interfaces for display of information (both input and inferred)**

Future Direction— GeoTemper



- Integration of the three dimensions of spatial knowledge with the temporal dimension to create a unified approach for handling change

